WEST Search History

DATE: Monday, February 03, 2003

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT,PGPB; PLUR=YES; OP=ADJ		
L10	(application adj4 service adj4 provider) same (add or adding) same component same application	6	L10
L9	(application adj4 service adj4 provider) same ((identify or identifying) near8 application) same (analyze or analyzing) same database	0	L9
L8	L7 and l6	1	L8
L7	spreadsheet near8 database	1650	L7
L6	L5 and l4 and l3	12	L6
L5	L2 and (operation adj5 architecture)	130	L5
L4	L2 and (development adj5 architecture)	69	L4
L3	L2 and (execution adj5 architecture)	45	L3
L2	architecture near6 generation	1162	L2
L1	(decision adj3 support) same collaboration same (Knowledge adj3 management) same intergration	0	L1

END OF SEARCH HISTORY

L6: Entry 1 of 12 File: USPT Dec 31, 2002

DOCUMENT-IDENTIFIER: US 6502213 B1
TITLE: System, method, and article of manufacture for a polymorphic exception handler in environment services patterns

Brief Summary Text (2):
This application is related to U.S. Patent Applications entitled A SYSTEM, METHOD AND ARTICLE OF MANUFACTURE FOR A DEVELOPMENT ARCHITECTURE FRAMEWORK and A SYSTEM, METHOD AND ARTICLE OF MANUFACTURE FOR MAINTENANCE AND ADMINISTRATION IN AN E-COMMERCE APPLICATION FRAMEWORK, both of which are filed concurrently herewith and which are incorporated by reference in their entirety.

Drawing Description Text (52):
FIG. 50 portrays of a development architecture with a seamless integration of tools which can be plugged in for the capture and communication of particular deliverables;

Detailed Description Text (48):
Execution architecture 302

Detailed Description Text (49):
The execution architecture is a unified collection of run-time technology services, control structures, and supporting infrastructure upon which application software runs.

Detailed Description Text (51):
Development Architecture Framework 304

Detailed Description Text (52):
The Development Architecture Framework (DAF) is a unified collection of technology services, tools, techniques, and standards for constructing and maintaining application software.

Detailed Description Text (54):
Refer to the Development Architecture Framework application (referenced above) for more information.

Detailed Description Text (55):
Operations architecture 306

Detailed Description Text (56):
A unified collection of technology services, tools, standards and controls required to keep a business application production or development environment operating at the designed service level. It differs from an execution architecture in that its primary users are system administrators and production support personnel.

Detailed Description Text (84):
The following quote from a recent study of Large Complex Systems (LCS) stress the importance of a stable architectures in large systems: Successful delivery of an LCS solution depends on the early definition and use of common data applications and technology architecture. There is a high failure rate when the architecture is not defined, stabilized, and delivered early in an LCS effort. All significant LCS efforts involved the use of common or shared architectures. A successful effort, however, depended on early definition and delivery of a stable common architecture. Significant changes to the data, application, or technology architectures had severe negative effects on the timeliness of project deliverables, and on the reliability of what was delivered. PROJECT1 and PROJECT2, for example, experienced unusual circumstances. While the client evaluated whether to proceed, one defines and designs the architecture. As a result, the teams had nine months to define, design, and begin implementation of

required data, applications, and development architectures. Although in each case these architectures continued to evolve with business and technology needs, they remained largely consistent with the initial design. This consistency proved to be essential to the timely delivery of the applications. At PROJECT3 and PROJECT4, on the other hand, the architectures went through major evolutions as the developers created the applications. The overall result was that those efforts experienced delays relative to plan. Although it is not realistic for every project to have nine months to define required architectures, it does suggest that early focus on definition and design of the architectural components is essential. The risk of failure is greatly increased if essential architectures are being defined or changed significantly in parallel with application development.

Detailed Description Text (95):
A Delivery Vehicle is an integrated collection of technology services that supports an application style, implemented on a distinct architecture generation.

Detailed Description Text (100):
Architecture generation

Detailed Description Text (101):
An architecture generation is a broad classification scheme for placing technology components within a technology era. Delivery Vehicles are physically implemented on a distinct architecture generation. Examples of architecture generations include host-based, client-server and netcentric.

Detailed Description Text (110):
Both the core services and the delivery vehicle extensions require support in all three environments. The cube illustrates that different delivery vehicles may require different extensions to a core development or operations environment, not just the execution architecture. A mission-critical high-volume transaction delivery vehicle may require special performance tuning tools in the development architecture, as well as real-time monitoring tools in the operations architecture.

Detailed Description Text (111):
Also different technology generations may require special services in all three environments. When working in a multi-platform environment, there may be duplicated services across platforms. This usually complicates development, operations and execution architectures and may require special focus on providing an integration architecture.

Detailed Description Text (113):
Typically, one may focus on engagements regarding the execution environment. The main dependency between these three environments is that the execution architecture to a large degree drives the requirements for the development and operations architectures. For example if a heterogeneous, distributed execution architecture is selected, both the development and operations environments must reflect this.

Detailed Description Text (128):
It is important to realize that a distinct, static division does not exist between the different technology generations. It is possible that an architecture may consist of components from more than one generation.

Detailed Description Text (132):
From a technology point of view a new custom-made application should generally use the most recent Architecture Generation to assure that the application will live longer by better being able to adapt to future changes.

Detailed Description Text (140):
Network centric architecture generation

Detailed Description Text (146):
The ability to digitize, organize, and deliver textual, graphical and other information (e.g., video, audio, etc.) in addition to traditional data to a broader audience, enables new methods for people and enterprises to work together. Netcentric technologies (e.g., HTML documents, plug-ins, Java, etc.) and standardization of media information formats enable support for these types of complex documents and applications. Network bandwidth remains a performance issue. However advances in network technologies and compression techniques continue to make richer media-enabled documents and applications more feasible on the Web. G4. The Execution, Operation and

Development architectures will be designed to support frequent releases of enhancements/modifications to production applications. It is imperative that companies in the current market place be able to quickly modify their business processes in order to address changes in the industry. A Netcentric architecture simplifies frequent software releases for both internal and external users of the systems.

Detailed Description Text (152):
IT guiding principles 804 G1. The client maintains their applications internally and the IT department has the necessary resources, organizations and processes to maintain a Client Server application. Introduction of a Client Server application to a company's production environment can require a great deal of change to the Execution, Operations and Development architectures required to develop, run and support the production systems. Before a Client Server application is developed, it is important that the client identify how a system of this type will fit within the company's strategic technology plan.

Detailed Description Text (153):
Host architecture generation

Detailed Description Text (163):
SAF provides access to the user's thought leadership and architecture frameworks for Execution, Development and Operations environments. Very briefly, SAF covers: The Core Execution Architecture frameworks for the different architecture generations (Host, Client/Server and Netcentric). Most users will primarily use the Netcentric framework. The Execution Architecture Extensions. This is a collection of the most common delivery vehicles that are built for clients. These frameworks extend the core frameworks with services specific for a particular delivery vehicle. The Development Architecture Framework. Should help one establish and operate a high-quality development environment. The Operations Architecture Framework. Should help one establish and operate a high-quality operations environment. To learn more about what Delivery Vehicles are, see the Delivery Vehicle Overview section. This page explains the relationships between Architecture Generations, Application Styles and Environments.

Detailed Description Text (171):
This framework constitutes the core of a modern netcentric and client/server execution architecture. It will help one plan and design one's architecture by understanding what components a typical netcentric architecture should consist of

Detailed Description Text (174):
The Netcentric Architecture Framework identifies those run-time services required when an application executes in a Netcentric environment. As shown in FIG. 10, the services can be broken down into logical areas: Presentation Services 1000, Information Services 1002,1004, Communication Services 1006,1008, Communication Fabric Services 1010, Transaction Services 1012,1014, Environment Services 1016,1018, Base Services 1020 and Business Logic 1022,1024. This framework is an evolution of the Client Server New Age Systems Framework and is useful for technical architects involved in the selection, development and deployment of technical architectures in a Netcentric environment. More discussion of each of these logical areas is provided below. See also FIGS. 11 and 12, which are detailed diagrams of the components of the Netcentric Architecture Framework found in FIG. 10.

Detailed Description Text (370):
Synchronization Services perform the transactions required to make one or more information sources that are intended to mirror each other consistent. This function may especially valuable when implementing applications for users of mobile devices because it allows a working copy of data or documents to be available locally without a constant network attachment. The emergence of applications that allow teams to collaborate and share knowledge has heightened the need for Synchronization Services in the execution architecture.

Detailed Description Text (744):
The following are examples of ways to implement Authorization services: Network Operating Systems--Authorization services are bundled with all network operating systems in order to control user access to network resources. Firewall Services protect sensitive resources and information attached to an Intxxnet network from unauthorized access by enforcing an access control policy. A variety of mechanisms exist for protecting private networks including: Filters--World Wide Web filters can prevent users from accessing specified content or Internet addresses. Products can limit access based on keywords, network addresses, time-of-day, user categories, etc. Application

Proxies--An application-level proxy, or application-level gateway, is a robust type of firewall. (A firewall is a system that enforces an access control policy between a trusted internal network and an untrusted external network.) The application proxy acts at the application level, rather than the network level. The proxy acts as a go-between for the end-user by completing the user-requested tasks on its own and then transferring the information to the user. The proxy manages a database of allowed user actions, which it checks prior to performing the request. Servers, Applications, and Databases--Authorization can occur locally on a server to limit access to specific system resources or files. Applications and databases can also authorize users for specific levels of access within their control. (This functionality is within the Environment Services grouping in the execution architecture.)

Detailed Description Text (937):
Environment Verification Services ensure functionality by monitoring, identifying and validating environment integrity prior and during program execution. (e.g., free disk space, monitor resolution, correct version). These services are invoked when an application begins processing or when a component is called. Applications can use these services to verify that the correct versions of required Execution Architecture components and other application components are available.

Detailed Description Text (957):
Primarily there are three types of errors: system, architecture and application. System errors occur when the application is being executed and some kind of serious system-level incompatibility is encountered, such as memory/resource depletion, database access problems, network problems or printer related problems, because of which the application cannot proceed with its normal execution. Architecture errors are those which occur during the normal execution of the application and are generated in architecture functions that are built by a project architecture team to isolate the developers from complex coding, to streamline the development effort by re-using common services, etc. These architecture functions perform services such as database calls, state management, etc. Application errors are also those which occur during the normal execution of the application and are generally related to business logic errors such as invalid date, invalid price, etc.

Detailed Description Text (1040):
The report architecture within Environment Services supports the generation and delivery of reports. Applications request report services by sending a message to the reporting framework.

Detailed Description Text (1124):
The execution architecture services are all generalized services designed to support the applications Business Logic. How Business Logic is to be organized is not within the scope of the execution architecture and must be determined based upon the characteristics of the application system to be developed. This section is intended to serve as a reminder of the importance of consciously designing a structure for Business Logic which helps to isolate the impacts of change, and to point out that the underlying Netcentric architecture is particularly well suited for enabling the packaging of Business Logic as components.

Detailed Description Text (1167):
The Management Considerations section discusses the key benefits, risks, and issues introduced by a component engagement. Key topics include: Managing risk in balancing tradeoffs between strategy, people, process, and technology Considering issues related to configuration management, testing, and performance of object systems Addressing the component development learning curve Differences between development architecture considerations leveraging the advantages of a component industry.

Detailed Description Text (1302):
Component-based development should not be understood as just a technology decision; rather, it is a new approach for software engineering. Thus, it affects almost all aspects of development including methodology, tools, organization, and architecture approaches. This broad impact creates multiple learning curves, complicating the migration of an organization. Finding available skills is also difficult, because demand currently outweighs supply.

Detailed Description Text (1303):
Component-based systems may also require immature technology or tools. Many of the core development tools such as the programming language and environments for C++, Visual Basic, Java and Smalltalk are actually very robust. However, some of the ancillary

tools such as the CASE tools and web development tools or technology architecture components such as messaging middleware may not be as mature. Thus, the team may face a choice of managing some risk exposure with a tool or library that simplifies development, or avoiding this tool risk but facing a more complex development challenge.

Detailed Description Text (1329):
As with client/server, architecture work must start early. As noted above, this is particularly challenging because of the level of application reuse in a well-designed application framework and domain component model. Because of this reuse, the framework must be heavily driven by application requirements, or scenarios. Yet, the architecture team must stay one step ahead of application development teams to ensure that the architecture and component model are ready in time to be reused. Thus, a difficult tension exists between scenarios and frameworks.

Detailed Description Text (1330):
The tension between scenarios and frameworks can be simplified to the extent that third-party or standard architectures such as Eagle can be leveraged. In any case, the following guidelines should be considered, particularly for custom architectures: The architecture should be defined and prototyped, if necessary, early in the preliminary design The architecture should be complete-at the very least, the development architecture and overall framework, prior to developers actually coding; the design must be in place earlier when functional developers start detailed design; private architecture aspects may be deferred Time must be planned for architecture support based upon unforeseen scenarios, performance tuning, documentation and developer mentoring Developing a custom application framework should be estimated as a set of tasks in addition to much of the traditional technology architecture development

Detailed Description Text (1395):
Avoid starting inexperienced people in architecture roles. There are simply too many skills to learn. Architects need to have a deep knowledge of design patterns, programming languages, technical infrastructure, and methodologies. It is better to start new developers in application development roles where they may have the opportunity to view the architecture as a consumer. This perspective may make them more effective in future architecture roles.

Detailed Description Text (1398):
This type of development approach requires a strong architecture vision that is clearly communicated and supported through training, mentoring, and documentation. If a strong vision does not exist, then the components may inevitably not fit together into a cohesive, integrated architecture. In addition, this strong vision must include an understanding of the business objectives and functions of the system to be effective.

Detailed Description Text (1465):
Architecture roles must be defined to support this greater degree of specialization. One engagement used the following partitioning strategy: Functional architect-responsible for resolving decisions for what the system should do. This person is ideally a user with a solid understanding of systems, or a systems person with a good understanding of, and relationship with, the users. Technology architect-responsible for delivering the platform, systems software, and middleware infrastructure to support execution, development, and operations architectures. User interface architect-responsible for setting direction of the user interface metaphor, layout standards, and integrated performance support (IPS). Application frameworks architect-responsible for designing, delivering, and supporting the application framework that provides the overall structure, or template, of the application. Object model architect-responsible for identifying and resolving modeling issues necessary to achieve a high degree of business reuse and modeling consistency.

Detailed Description Text (1496):
The higher degree of custom development required in the architecture, the more specialization of skills is necessary; likewise, the more stable the architecture, the less important is specialization in favor of supporting collaboration

Detailed Description Text (1559):
As with client/server, architecture work must start early. As noted above, this is particularly challenging because of the level of application reuse in a well-designed application framework and domain component model. Because of this reuse, the framework must be heavily driven by application requirements, or use cases. Yet, the architecture team must stay one step ahead of application development teams to ensure that the

architecture and component model are ready in time to be reused. Thus, a difficult tension exists between use cases and frameworks.

Detailed Description Text (1560):
The tension between use cases and frameworks can be simplified to the extent that third-party or standard architectures such as Eagle can be leveraged. In addition, experienced architects may bring their knowledge of which services are common across applications and can be addressed earlier than application-specific architecture services. In any case, the following guidelines should be considered, particularly for custom architectures: The architecture should be defined and prototyped, if necessary, early in the preliminary design The architecture should be complete-at the very least, the development architecture and overall framework, prior to developers actually coding; the design must be in place earlier when functional developers start detailed design; private architecture aspects may be deferred Time must be planned for architecture support based upon unforeseen use cases, performance tuning, documentation and developer mentoring Developing a custom application framework should be estimated as a set of tasks in addition to much of the traditional technology architecture development

Detailed Description Text (1669):
Development Architecture Considerations

Detailed Description Text (1670):
This section highlights key messages for development architecture teams in regard to supporting teams and tools within a component based development project.

Detailed Description Text (1671):
Building systems that are dramatically more responsive to change require a dramatically improved development architecture.

Detailed Description Text (1680):
Ideally what would greatly increase the productivity of the development architecture is a seamless integration of tools in the workbench and the ability to "plug in" whatever tool is most appropriate for the capture and communication of a particular deliverable. FIG. 50 portrays a development architecture with a seamless integration of tools which can be plugged in for the capture and communication of particular deliverables. Shown in FIG. 50 is the relationship between a process phase 5000, deliverables 5002, tools 5004, repositories 5006, and an information model 5008.

Detailed Description Text (1684):
A development architecture should provide an environment for component-based solutions that supports a team through the Analysis, Design, and Construction phases of the development process. It should also serve as a productive environment for the on-going maintenance of an application. Conceptually it should integrate all of the necessary tools through an information model and most ideally through a central repository. The following are considerations that all component development architecture must consider. 1. Support Custom Process. The present invention uses a robust process for developing component-based solutions. It includes deliverables that are above and beyond the Unified Modeling Language (UML). Furthermore, projects often customize it. The environment must provide the ability to extend the information model (i.e., the meta-model). 2. Versioning & configuration management. The environment should provide the ability to version objects within the common information model at any level of granularity, keeping track of these changes over time. It should provide the same ability for composite objects (i.e., configurations of smaller objects). 3. Scalability. The repository-enabled environment must be able to support hundreds of users simultaneously, and hundreds of thousands of repository relationships. It should also scale downward, so that small project can use it. This is a major criterion for usability. 4. Query and impact analysis. As organizations begin to maintain their own component-based assets, they must be able to analyze the impact of change requests (e.g., where-used searches). The ability to trace requirements is also critical. 5. Asset catalog (reuse). As organizations begin to reuse existing assets, it may become increasingly important to provide a catalog of components, frameworks, patterns, etc. The catalog should make it possible to search for relevant assets in a wide variety of ways. It should also provide a means for applying frameworks and patterns. 6. Code generation. The ability to generate the application structure from the model is essential to high productivity. Furthermore, this step should be transparent to the user. As far as the user is concerned, a change to the model is a change to the code. 7. Desktop Tool Integration. The repository-enabled environment must provide integration between all desktop tools (e.g., MS Office, Visio, OO CASE tools,

designers, etc.) through component object models such as ActiveX. In addition, these tools must have access to the common open information models. 8. Non-redundant storage. The environment should avoid redundant storage of information, whenever possible. Everything from training to documentation to active components should be automatically updated or notified of changes. 9. Multiple users and locations. Many users may need access to the environment during the course of a development effort. Furthermore, because one supports global communities of practice, there is a strong need to share this information securely and across disparate locations.

Detailed Description Text (1685):
A Development Architecture Needs to Support Customization of the Process.

Detailed Description Text (1686):
UML & Case Tools in the development architecture

Detailed Description Text (1690):
Case tools in recent years have extended their ability to support more of the life cycle and improved their ease of use. In addition, some case tools have improved their integration with the Integrated Development Environments (IDEs) and produce some level of acceptable component code generation. It is important for the development architecture team to determine early exactly which deliverables may be created in each phase of development, which tool they may be captured in and whether links between phases require upgrading deliverables as a result of the transformations and/or enhancements from other phases.

Detailed Description Text (1692):
Development Architectures Are Often More Heterogeneous Than Traditional Environments

Detailed Description Text (1694):
Typically, the more heterogeneous environments may be built upon the open CORBA technology, while applications developed with JavaBeans or COM may tend to be more homogeneous in nature. Thus, it is important to understand the technologies used as the effort to design a cohesive development architecture may be impacted. Plan to spend more time designing and building the development architecture for a heterogeneous environment.

Detailed Description Text (1697):
Configuration management is more complex in a component development architecture

Detailed Description Text (1747):
At least 5% of the development team should be completely dedicated to the on-going configuration management effort. When setting up and defining the environment even more resources may be necessary. Of course, there are limits. Stacking the team with too many resources may result in wasteful development of an overly elaborate tools architecture.

Detailed Description Text (1750):
Despite the advice to use small teams, enterprise applications are large and often require in the aggregate a large number of developers. Development architectures must be constructed in such a way as to support sometimes hundreds of users with many, sometimes hundreds of thousands of development artifacts and their relationships with each other.

Detailed Description Text (1752):
The new strategies all espouse either de facto standards (Microsoft's Open Information Model) or eventual conformance to a repository strategy (OMG's Meta Object Facility--MOF). These repositories, although encouraging, are very immature and may require a few years to deliver on their promises. In the mean time development architectures must decide on their own how they may provide the necessary facilities to promote large team development progress.

Detailed Description Text (1756):
One of the problems in current development architectures is the redundancy of the facilities. For example, rather than be able to rely on the repository where the information should be stored in a common location developers may search in Rational Rose and in the source code manager for references of a given type.

Detailed Description Text (1768):
ODM has many predefined deliverable templates that are targeted towards this suite of

tools including Word, Excel and Visio templates. Often times management underestimates the start up cost of integrating the tools in such a way as to improve the flow of information between phases and for ensuring that information is published to the team in a way that is accessible and plentiful. However project experience teaches that this investment can yield many returns down the road if the development architecture includes processes and infrastructure to support this flow of information.

Detailed Description Text (1771):
Solution Centers and engagements often have many users and multiple locations involved in solution delivery. It is very important for development architecture teams to solve the problems of concurrency within tools and ownership across locations. Strategies need to be developed for how components may be exported and imported, and supported across locations. In addition, an approach for receiving feedback for improvements needs to be established. Most projects have found that ownership is even more important in a distributed development environment. This allows for the using of master/slave assignments on components and dictating either who is allowed to make changes to the component or who is responsible for merging changes. As one technologist from Sun stated, if distributed development is not managed carefully it can be like herding cats.

Detailed Description Text (1773):
Although there are new challenges with development architecture in a component environment there are also additional opportunities for increased productivity. A team that understands the additional considerations may weigh the opportunities that tool integration can bring to the project against the practical gap in the market place and customize their development architecture accordingly. Wise planning and a clear understanding of the strengths and limitations of the tools available to a team may contribute greatly to the success or failure of a project.

Detailed Description Text (2600):
The patterns in this section solve several of the fundamental problems encountered in the development of an object-to-relational persistence architecture, including the mapping of classes to tables (Data Handler, Individual Persistence), identity management (Object Identifiers as Object), caching(Object Identity Cache), allocation of responsibilities (Data Handler, Piecemeal Retrieval, Persistent State Separate from Persistent Object), and data access optimization (Multi-Object Fetch) and the mapping of basic SQL types to object attributes (Attribute Converter).

Detailed Description Text (2752):
From a persistence perspective, no matter which of these approaches is used, to development of the system and architecture presents two distinct challenges to the development team. The first challenge is to accurately represent the business logic as a collection of business objects that include interfaces for performing the correct set of functionality. The second challenge is to be able to create, retrieve, update and delete (CRUD) records that represent the state of these business objects from the database in an efficient fashion.